UNITED STATES

PATENT APPLICATION

for

METHOD AND SYSTEM FOR EXECUTING DATABASE QUERIES

NCR Docket No. 11178

submitted by

Ahmad Ghazal

on behalf of

Teradata
a Division of NCR Corporation
Dayton, Ohio

Prepared by

Michael A. Hawes
Reg. 38,487

Correspond with

John D. Gowart
Reg. 38,415
Teradata Law IP, WHQ-4W
NCR Corporation
1700 S. Patterson Blvd.
Dayton, OH 45479-0001
(858) 485-4903 [Voice]
(858) 485-2581 [Fax]

Express Mail No.: EV339222797US
Date: October 29, 2003

Method and System for Executing Database Queries

Background

[0001] Data stored in database systems often is accessed by users who provide queries defining the type of information that they would like to receive. For example, a query in a database that contains employee information could define a range of dates and request information on all employees who started to work for the company in the specified time range. Often, there are a large number of possible approaches that will each produce the information requested by the query. Each approach involves a series of database access steps. For example, a query seeking the last names of all employees who sold more than 100 products in a particular month could require information from different tables stored in the database. One table that includes a record for each product sold including the employee number of the person that sold it could be used to determine the employee numbers with more than 100 sales. Another table could be used to determine the last name that corresponds to each of those employee numbers. The time and system resources expended to provide the query answer can depend on the order in which those steps are performed.

[0002] One conventional way to choose the order in which steps will be performed to execute a database query includes estimating the time and system resources that will be necessary for each possible series of steps. The series of steps with the lowest cost can then be selected. Data to be retrieved or manipulated by a step is often chosen in accordance with conditions. For example, retrieving data regarding a certain employee might involve using a condition that the employee_number field be equal to the employee's actual number. Other examples of conditions include a comparison condition, e.g., all employees who started before a particular date, or a range condition, e.g., all products costing between $19 and $99. The database system can determine the conditions corresponding to a step based on the query and the analysis of that query. The database system employees those conditions both to estimate the cost of a particular series of steps and to actually carry out the series of steps chosen to execute the query.

Summary

[0003] In general, in one aspect, the invention features a method for executing database queries. The method includes identifying a first set of conditions corresponding to a selected step for executing a query. A second set of conditions corresponding to one or more steps for executing the query that feed the selected step is identified. Each condition in the first set is checked for mathematical redundancy, including redundancy without equivalency, with regard to the other conditions in the union of the conditions corresponding to the selected step and the conditions in the second set. Each condition in the first set that is redundant is included in a third set. If there is only one condition in the third set, an identifier of the one condition is stored.

[0004] Implementations of the invention may include one or more of the following. An estimate of the cost of performing the steps can be performed without taking into account any identified conditions. When the third set contains multiple conditions, the conditions can be sorted with only the initial condition being identified. Once a condition is identified, redundancy can be checked for the remaining conditions without the identified condition. The query can be executed without evaluating any of the identified conditions.

[0005] In general, in another aspect, the invention features a computer program for executing database inquires. The program includes executable instructions that cause a computer to identify a first set of conditions corresponding to a selected step for executing a query. A second set of conditions corresponding to one or more steps for executing the query that feed the selected step is identified. Each condition in the first set is checked for mathematical redundancy, including redundancy without equivalency, with regard to the other conditions in the union of the conditions corresponding to the selected step and the conditions in the second set. Each condition in the first set that is redundant is included in a third set. If there is only one condition in the third set, an identifier of the one condition is stored.

[0006] In general, in another aspect, the invention features a database system for executing database queries. The database system includes one or more nodes; a plurality of CPUs, each of the one or more nodes providing access to one or more CPUs; and a plurality of virtual processes, each of the one or more CPUs providing access to one or more virtual processes, each virtual process configured to manage data, including rows organized in tables, stored in one of a

plurality of data-storage facilities. The database system also includes an optimizer that is configured to identify a first set of conditions corresponding to a selected step for executing a query. A second set of conditions corresponding to one or more steps for executing the query that feed the selected step is identified. Each condition in the first set is checked for mathematical redundancy, including redundancy without equivalency, with regard to the other conditions in the union of the conditions corresponding to the selected step and the conditions in the second set. Each condition in the first set that is redundant is included in a third set. If there is only one condition in the third set, an identifier of the one condition is stored.

Brief Description of the Drawings

[0007] Fig. 1 is a block diagram of a node of a parallel processing database system.

[0008] Fig. 2 is a block diagram of a parsing engine.

[0009] Fig. 3 is a flow chart of one method for estimating the cost of a database query execution plan.

[0010] Fig. 4 is a flow chart of one method for executing a database query execution plan.

[0011] Fig. 5 is a flow chart of one method of identifying conditions for a query execution step and its feed steps.

[0012] Fig. 6 is a flow chart of one method of identifying optimal redundant conditions for a query execution step.

[0013] Fig. 7 is a flow chart of one method of sorting redundant conditions corresponding to a query execution step.

[0014] Fig. 8 is a flow chart of one method of identifying optimal redundant conditions for a query execution step that are CPU redundant.

Detailed Description

[0015] The query execution technique disclosed herein has particular application, but is not limited, to large databases that might contain many millions or billions of records managed by the database system ("DBS") 100, such as a Teradata Active Data Warehousing System available from NCR Corporation. Figure 1 shows a sample architecture for one node 105_1 of the DBS 100. The DBS node 105_1 includes one or more processing modules $110_{1...N}$, connected by a network 115, that manage the storage and retrieval of data in data-storage facilities $120_{1...N}$. Each of the processing modules $110_{1...N}$ may be one or more physical processors or each may be a virtual processor, with one or more virtual processors running on one or more physical processors.

[0016] For the case in which one or more virtual processors are running on a single physical processor, the single physical processor swaps between the set of N virtual processors.

[0017] For the case in which N virtual processors are running on an M-processor node, the node's operating system schedules the N virtual processors to run on its set of M physical processors. If there are 4 virtual processors and 4 physical processors, then typically each virtual processor would run on its own physical processor. If there are 8 virtual processors and 4 physical processors, the operating system would schedule the 8 virtual processors against the 4 physical processors, in which case swapping of the virtual processors would occur.

[0018] Each of the processing modules $110_{1...N}$ manages a portion of a database that is stored in a corresponding one of the data-storage facilities $120_{1...N}$. Each of the data-storage facilities $120_{1...N}$ includes one or more disk drives. The DBS may include multiple nodes $105_{2...P}$ in addition to the illustrated node 105_1, connected by extending the network 115.

[0019] The system stores data in one or more tables in the data-storage facilities $120_{1...N}$. The rows $125_{1...Z}$ of the tables are stored across multiple data-storage facilities $120_{1...N}$ to ensure that the system workload is distributed evenly across the processing modules $110_{1...N}$. A parsing engine 130 organizes the storage of data and the distribution of table rows $125_{1...Z}$ among the processing modules $110_{1...N}$. The parsing engine 130 also coordinates the retrieval of data from the data-storage facilities $120_{1...N}$ in response to queries received from a user at a mainframe 135 or a client computer 140. The DBS 100 usually receives queries and commands to build tables in a standard format, such as SQL.

[0020] In one implementation, the rows $125_{1...Z}$ are distributed across the data-storage facilities $120_{1...N}$ by the parsing engine 130 in accordance with their primary index. The primary index defines the columns of the rows that are used for calculating a hash value. The function that produces the hash value from the values in the columns specified by the primary index is called the hash function. Some portion, possibly the entirety, of the hash value is designated a "hash bucket". The hash buckets are assigned to data-storage facilities $120_{1...N}$ and associated

processing modules $110_{1...N}$ by a hash bucket map. The characteristics of the columns chosen for the primary index determine how evenly the rows are distributed.

[0021] Figure 2 components of the parsing engine 130. An SQL request 210 is submitted to the parsing engine 130 and is initially checked for syntax 220. The resolver 230 then checks for and reports semantic errors and determines additional conditions based on transitivity. If one condition requires that the price is $10 and another requires that the cost is half the price, a third condition can be determined by transitivity: the cost is $5. The new conditions can be redundant with the original conditions, but can result in faster execution. For example, it is possible for a query to run more quickly with conditions of price=$10 and cost=$5 than with conditions of price=$10 and cost=50%(price).

[0022] Once the query has been processed by the resolver 230, it is passed to the security component 240 of the parsing engine 130. The security component 240 checks the security level of the database user who initiated the query. The security component 240 also checks the security level of the information sought by the request. If the user's security level is less than the security level of the information sought, then the query is not executed.

[0023] Once the query passes security it is analyzed by the optimizer 250. The optimizer 250 determines possible series of steps for executing the query. The optimizer 250 also estimates the costs associated with each series of steps. The cost associated with a series of steps is related to the amount of data encompassed by each condition corresponding to a step in the series. The execution of a query involves temporary results and sub-query results and the amount of data in those results is one factor in determining the costs of executing the query. The cardinality of temporary results or sub-query results refers to the cost. A temporary result that requires a large amount of system resources to generate has high cardinality.

[0024] After estimating the costs associated with potential query execution plans, the optimizer 250 chooses the plan that has the lowest estimated cost. The more accurate the estimates of cardinality for particular execution plans, the more likely the optimizer 250 is to choose the correct plan. The optimizer 250 can access statistics describing the information stored in the database to help estimate the cardinality of conditions and temporary results corresponding to steps in query execution plans.

[0025] The plan chosen by the optimizer 250 is passed to the step generator 260. The steps are then sent to the step packager 270 and dispatched from the step dispatcher 280. If the plan chosen is not the optimal plan, the steps generated will require the use of more resources than the steps that would be generated by another plan that yields the same output. In a parallel database system servicing thousands of concurrent users, an increase in the resources employed for reach query can result in longer wait times for every user.

[0026] Figure 3 is a flow chart of one method for estimating the cardinality of non-final results for an execution plan 300. First, ordered execution steps for a potential database query execution plan are generated 310. The conditions corresponding to each step are determined 320. Conditions that are optimal redundant for a particular step are determined 330. Figures 5 and 6 discuss the process of determining optimal redundancy in more detail. When several conditions are individually redundant, but are not all collectively redundant, the optimally redundant condition(s) are those that result in the most accurate cardinality estimation when removed from the estimation process. An optimal redundant condition can also be referred to as a selectivity redundant condition. For example, if a query step includes three conditions: price=$10; cost=$5; and cost=50%(price), each of the three is redundant with regard to the other two. Only one can be removed, however, without losing information. Once the one or more conditions that are optimally redundant are determined, the cardinalities can be estimated without taking into account the optimal redundant conditions.

[0027] Figure 4 is a flow chart of one method for executing a query 400. First, ordered execution steps for an actual database query execution plan are generated 410. The conditions corresponding to each step are determined 420. Conditions that are optimal redundant for a particular step are determined 330. Figures 5 and 6 illustrate the process of determining optimal redundancy in more detail. Conditions that are CPU redundant as well as being optimal redundant are determined 430. Figure 7 illustrates the process of determining CPU redundancy is more detail. Once the one or more conditions that are CPU redundant are determined, the query can be executed without evaluating those conditions.

[0028] Figure 5 is a flow chart of one method of identifying conditions for a query execution step and its feed steps 330. First, one step is chosen and the conditions corresponding to that step are identified 500. All the steps that feed into the particular step are then identified 510 along

with the conditions that correspond to those steps 520. From the conditions corresponding to the particular step, the optimal redundant conditions are picked out 530. Figure 6 illustrates the process of determining optimal redundant conditions for a particular step in more detail. If there are more steps in the potential or actual execution plan 540, another step is chosen 500. Otherwise, all of the optimal redundant conditions for the steps have been identified.

[0029] Figure 6 is a flow chart of one method of identifying optimal redundant conditions for a query execution step 530. One of the conditions that has not been marked as optimal redundant is chosen 600 to be evaluated. All other conditions from the current step and all steps that feed it are combined in a union 605, except that conditions already marked as optimal redundant are excluded 610. The condition being evaluated is checked to see if it is redundant, including redundancy other than equivalency, with the union of conditions and it is included in a set if it is redundant 615. In one embodiment, a transitive closure function is performed on the union to determine whether the condition being evaluated results. If there are additional unmarked conditions 620, the process of picking out the redundant conditions is repeated 620. Once all of the redundant conditions are included in the set, the number of conditions in the set becomes important. If no conditions were redundant 625, then there are no optimal redundant conditions 635. If there is one condition in the set 625, that condition is marked as optimal redundant (or selectivity redundant) 630 and no more conditions are evaluated 635. If more than one condition is redundant 625, the conditions are sorted 640. The sorting step is discussed in more detail in Figure 7. Once sorted, the initial condition is marked as optimal redundant (selectivity redundant) 645. The condition loop is then reset 650 so that all the unmarked conditions will be checked for redundancy again in view of the newly marked conditions in steps 600-620. As fewer unmarked conditions corresponding to the step remain, the method will eventually result in no redundant conditions or only one.

[0030] Figure 7 is a flow chart of one method of sorting redundant conditions corresponding to a query execution step 640. The sorting function sorts redundant conditions according to their importance in relation to selectivity. The ordering method is based on statistics, i.e. if the columns in the condition have collected statistics on them or not. A simple count of the number of columns that have collected statistics can be used 700. The conditions are then sorted in order of that number 710. In case of a tie 720, the condition that was produced by transitive closure

will have the high sort order 730. If two conditions tie in both measures then their ordering can be either way.

[0031] Figure 8 is a flow chart of one method of identifying optimal redundant conditions for a query execution step that are CPU redundant 430. Ideally, to find out if a mathematically redundant condition is also CPU redundant the binary join costing is computed with and without that condition. If it less costly without the condition then the condition is CPU redundant. If this method is used and more than one CPU redundant condition is found then the cost could be used as the sorting criteria. This method is not practical because it is costly (from optimizer 250 point of view) to compute the cost of binary join planning. Also, the binary join costing may not be accurate in terms of the CPU cost of applying a condition. That is why heuristics will be to identify CPU redundant conditions and also rank them in terms of overhead.

[0032] The heuristics used to find if an optimal redundant condition is also CPU redundant are the following. If the condition is a single table condition, then it is CPU redundant only if none of the following two qualifications are true. (1) The condition does not provide an access path to the table. Examples of access paths if the condition is PI (primary index) = constant which provides a single AMP, single hash access, SI (secondary index) = constant which provides access to the table through a secondary index. I (index) in a range comparison where the index is value ordered. Note that whether the condition provides an access path to the table can be determined by looking at how the binary join is done by the optimizer 250. (2) The condition does not reduce the size of a temporary result. For example, assume that a table named lineitem is hashed by a column named l_partkey and a table named ordertbl is hashed by a column named o_orderkey, consider the binary join between lineitem and ordertbl. Assume that the binary join method found by the optimizer 250 is to redistribute lineitem to join with ordertbl. The condition l_orderkey=4 is applied to lineitem prior to re-distribution. Note that l_orderkey=4 does not provide an access path (no indexes on l_orderkey) but still it is not CPU redundant. The reason is that it may filter out rows from lineitem prior to the join.

[0033] If the condition is a join condition, then it is CPU redundant if it is not useful in co-locating the sources of the join. For example, l_orderkey=o_orderkey is selectivity redundant because it can be derived using l_orderkey=4 and o_orderkey=4. If either l_orderkey or

o_orderkey is a primary index of its table, then that helps the optimizer 250 to minimize data movement (duplication or re-distribution). Also, the knowledge of primary index helps the optimizer 250 avoid unnecessary sorting. After the binary join is found whether the condition helped in co-location can be determined. For example, if lineitem or ordertbl are duplicated or both are redistributed then the condition was not useful in co-locating the tables.

[0034] A condition that was marked as optimal redundant is picked 800. Depending on whether the condition is a single table condition or a join condition 810, a different evaluation is applied. If a single table condition is not an equality with an index 820, is not a range of a value ordered index 830, and does not reduce temporary result size 840, then the condition is marked as CPU redundant 850. Otherwise, it is not so marked. If a join condition does not co-locate join sources 860, then it is marked CPU redundant 870. Otherwise, it is not so marked. In either case, all the optimal redundant conditions are evaluated 880.

[0035] The foregoing description of the embodiments of the invention has been presented for the purposes of illustration and description. It is not intended to be exhaustive or to limit the invention to the precise form disclosed. Many modifications and variations are possible in light of the above teaching. It is intended that the scope of the invention be limited not by this detailed description, but rather by the claims appended hereto.